A Human-centred Tangible approach to learning Computational Thinking

Tommaso Turchi
Human Centred Design Institute
Brunel University London
United Kingdom
tommaso.turchi@brunel.ac.uk

Alessio Malizia
Human Centred Design Institute
Brunel University London
United Kingdom
alessio.malizia@brunel.ac.uk

***Abstract*—Computational Thinking has recently become a focus of many teaching and research domains; it encapsulates those thinking skills integral to solving complex problems using a computer, thus being widely applicable in our society. It is influencing research across many disciplines and also coming into the limelight of education, mostly thanks to public initiatives such as the Hour of Code. In this paper we present our arguments for promoting Computational Thinking in education through the Human-centred paradigm of Tangible End-User Programming, namely by exploiting objects whose interactions with the physical environment are mapped to digital actions performed on the system.**

Keywords—Computational Thinking; Education; Tangible User Interface; End-User Programming

I. Introduction

Given how our entire society is increasingly surrounded by technology, learning to code is becoming an essential skill to master for the general public: from the amount of software involved in managing a flight, to the simple task of turning on the engine of your car, it is unmistakably clear how much we rely on software in every part of our lives. That being the case, people will always strive to play a more active role in their life, resulting in a clear overall heightened interest in coding: take for instance the Hour of Code[1], a successful global initiative involving millions of students of different ages starting with 4-year old, aiming at introducing coding skills to a wide and heterogeneous audience.

Coding skills are not just about programming though: they require quite an ability of problem solving, abstraction, and pattern recognition to name but a few: in a word, the so-called Computational Thinking skills. In the following, we highlight the role of Computational Thinking in Education and present our arguments for fostering its development through pairing End-User Programming with Tangible User Interfaces.

II. Computational Thinking and Education

In her seminal work [1], Wing introduced Computational Thinking (CT) as a set of thinking skills, habits, and approaches integral to solving complex problems using a computer, thus widely applicable in today's information society. It encompasses far more than just programming, representing a range of mental tools reflecting the fundamental principles and concepts of Computer Science, including abstracting and decomposing a problem, recognizing similar ones and being able to generalize their solutions. It shares many of its concepts, practices and perspectives with other subject areas taught in schools, such as science, mathematics, arts and engineering, making a strong case for its teaching in disciplines outside of Computer Science and right from kindergarten [2].

The echo of the discussion around the importance of teaching programming and computational thinking in school [3] resounded already in the adoption of new curricula by some European nations, such as England, where coding is mandatory in elementary schools from year 1[2], Finland, planning to introduce computational thinking in its curriculum from this year, and Estonia, having had coding as part of its curriculum since 2013[3]. Moreover, many other extra European countries such as Russia, South Africa, New Zealand, and Australia already have Computer Science as part of their K-12 curriculum [4].

The idea of enabling pupils to participate in computational thinking has been around for thirty years, and can be traced back to the work of Seymour Papert [5]; he first proposed the idea of children engaging in coding and developed the LOGO programming environment to enable them to do so.

Many other tools and programming environments followed and have been developed with the aim of promoting CT skills in K-12 education; the main principle guiding their development was the idea of "low floor, high ceiling", i.e. they enable any beginner to cross the threshold to create working programs easily (low floor), but are also powerful enough to satisfy the needs of more advanced users (high ceiling). Besides, effective CT supporting tools for children must have low floor and high ceiling, provide stepping stones with managed skills and challenges to get them from the "floor" to the "ceiling" (scaffold), enable transfer between different application contexts, support enquiry and be systemic and sustainable [6].

Graphical programming environments like Scratch, Alice, Game Maker and Kodu closely follow these principles on

[1] https://hourofcode.com

[2] https://www.gov.uk/government/publications/national-curriculum-in-england-computing-programmes-of-study

[3] http://www.sitra.fi/en/artikkelit/well-being/future-will-be-built-those-who-know-how-code

various degrees: they are relatively easy to use and allow novices to focus on designing and creating while avoiding the issues of the traditional murky and complicated programming syntax.

Tools and programming environments however need to support – and in turn be supported – by curricular activities such as game design and robotics, typically serving as a trigger for the iterative exploration of CT while motivating and engaging school children. These activities – along with many similar ones – should support what have been proposed to be the four pedagogical phases of learning to think computationally [2]: (1) *unplugged activities* are used to inspire students and enhance subject knowledge, and can be implemented without the use of computers, making abstract concepts both tangible and visible [7]; (2) *making* activities include playing or making stuff, taking them apart and wanting to do so to solve problems inspired by technology; (3) *tinkering* [8] supports learning of CT concepts by exploring them in a creative way, allowing students to use materials and tools to represent computational thinking; (4) *remixing* (or "*hacking*") involves critically looking at existing code, as well as practicing modifying it to suit new purposes.

Lastly, it is worth pointing out that the effectiveness of existing tools seems pretty unsettled with respect to the many facets of Computational Thinking: for instance, a 2008 study [9] involving 80 urban youth aged 8-18 reported learning of several CT elements through the use of Scratch in an after-school setting; nonetheless, the tool doesn't provide a mean of encapsulating functionalities into procedures and functions, somehow failing to tap into the abstraction skills. There is undoubtedly a need for new tools that foster CT skills specifically targeted to K-12 education, following the principles just described and guided by the most recent research on how children approach problem solving [10], [11].

III. Fostering Computational Thinking skills with Tangible End-User Programming

Fostering the development of CT skills has recently become a focus of the End-User Development (EUD) research community [12], whose aim is to allow end users (i.e. any computer user) to adapt software systems to their particular needs at hand; it strives to enable them to exploit some of the computational capabilities enjoyed by professional programmers, thus to perform their tasks more efficiently and effectively (e.g. task automation). Computational Thinking, therefore, seems the ideal skill set needed to help the EUD community to reach its aim of lowering programming barriers and fostering its spread.

At the same time, programming itself has proven to be an excellent way of developing CT skills [13] especially in K-12 education (as discussed in the previous section), thus existing EUD techniques might have a similar effect on end users, although much discussion is still ongoing about how much this comes about and results are still inconclusive.

Since its introduction, there have been many attempts of defining more precisely what Computational Thinking actually means [14]. Even though there is not yet an agreement on it, the only consensus reached so far between the different proposed definitions pertains to the concepts of abstraction and decomposition:

- Abstraction helps modeling problems and systems by capturing only the essential properties common to a set of objects while hiding their differences, the latter being not relevant from the perspective we are currently interested in.
- Decomposition refers to thinking about a problem (or – more generally – an artifact, e.g. a system, a process, or an algorithm) in terms of its components; one can then understand, solve and evaluate them separately, making the overall problem easier to solve.

These two concepts are an integral part of Computational Thinking, as well as frequently required during any programming task. Dealing with abstract concepts is often a challenge for inexperienced users, who usually need to be trained and practice this skill for quite a while before mastering it.

While developing his theories on learning and developing LOGO, Papert widely referred himself to the work of Jean Piaget: his constructivist theory [15] tries to explain how human capabilities evolve during the first years of life: at ages 7 to 11 children are in what he called *concrete operational stage*; they can think logically in terms of objects, but have difficulty replacing them with symbols. Ultimately, they can solve problems in a logical fashion, but are typically not able to think abstractly or hypothetically. The following stage, i.e. the *formal operational stage*, enables them to replace objects with symbols, generalizing and manipulating abstract concepts by using proportional reasoning and deriving cause-effect relationships. The shift from concrete operational to formal operational should occur by age 12, but a later study [16] found that most College freshmen in physics courses still haven't made it, being incapable of grasping abstract concepts not firmly embedded in their concrete experience.

In the light of these considerations, we argue that exploiting our innate dexterity for objects' manipulation in the physical world could be an effective way of aiding concrete operational thinkers to grasp abstract concepts often involved by coding, thus fostering the development of Computational Thinking skills. Physical manipulation is an interaction paradigm currently employed in digital systems with the aim of providing users with an easy to use interface that can be used even by inexperienced people; this paradigm is called Tangible User Interfaces (TUIs) [17]. Employing a TUI in an End-User Development system – thus pairing it up with a technique with the aim of lowering programming barriers and allowing end users to program – could foster their Computational Thinking skills by supporting them with a concrete representation of the abstract concepts they have to deal with.

In a recent study [18] we introduced TAPAS (TAngible Programmable Augmented Surface), a system that allows users to adapt a public display's features to their own needs, by using the movements of their smartphone to interact with it; users can develop simple workflows by assembling different services together by means of a puzzle: each service is mapped to a puzzle piece and its shape dictates constraints on its required

inputs and output. Even though components are not physically represented here but only digitally, the system is controlled through a tangible object, making it fun and easy to use. Nevertheless, further investigations are needed to see whether the proposed interaction modality helps developing users' Computational Thinking skills in conjunction with its application domains. We are currently running a study with high school students to investigate such implications in an educational domain.

Summarizing, we believe that exploiting the benefits of using a tangible interaction in conjunction with EUD techniques will leverage on human's natural ability of manipulating objects in the real world, aiding end users in grasping highly abstract concepts while fostering their Computational Thinking skills.

REFERENCES

[1] J. M. Wing, "Computational thinking," *Commun. ACM*, vol. 49, no. 3, p. 33, Mar. 2006.

[2] I. K. Namukasa, D. Kotsopoulos, L. Floyd, J. Weber, Y. Kafai, S. Khan, C. Yiu, L. Morrison, and S. Somanath, "From computational thinking to computational participation: Towards Achieving Excellence through Coding in elementary schools," 2015.

[3] *Report of a Workshop on the Pedagogical Aspects of Computational Thinking*. Washington, D.C.: National Academies Press, 2011.

[4] S. Grover and R. Pea, "Computational Thinking in K-12: A Review of the State of the Field," *Educational Researcher*, vol. 42, no. 1, pp. 38–43, Feb. 2013.

[5] S. Papert, *Mindstorms: children, computers, and powerful ideas*. Basic Books, Inc., 1980.

[6] A. Repenning, D. Webb, and A. Ioannidou, "Scalable game design and the development of a checklist for getting computational thinking into public schools," presented at the the 41st ACM technical symposium, New York, New York, USA, 2010, pp. 265–269.

[7] P. Curzon, "Cs4fn and computational thinking unplugged," presented at the ACM International Conference Proceeding Series, 2013, pp. 47–50.

[8] M. U. Bers, L. Flannery, E. R. Kazakoff, and A. Sullivan, "Computational thinking and tinkering: Exploration of an early childhood robotics curriculum," *CE*, vol. 72, pp. 145–157, Mar. 2014.

[9] J. Maloney, K. Peppier, Y. B. Kafai, M. Resnick, and N. Rusk, "Programming by choice: Urban youth learning programming with scratch," presented at the SIGCSE'08 - Proceedings of the 39th ACM Technical Symposium on Computer Science Education, 2008, pp. 367–371.

[10] T.-Y. Chen, G. Lewandowski, R. McCartney, K. Sanders, and B. Simon, "Commonsense computing," *SIGCSE Bull.*, vol. 39, no. 1, p. 276, Mar. 2007.

[11] J. F. Pane, C. A. Ratanamahatana, and B. A. Myers, "Studying the language and structure in non-programmers' solutions to programming problems," *International Journal of Human-Computer Studies*, vol. 54, no. 2, pp. 237–264, Feb. 2001.

[12] "Foreword VL/HCC 2015," *2015 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pp. vii–viii, 2015.

[13] G. Orr, "Computational thinking through programming and algorithmic art.," *SIGGRAPH Talks 2009*, pp. 1–1, 2009.

[14] C. Selby and J. Woollard, "Computational thinking: the developing definition," 2013.

[15] B. Inhelder and J. Piaget, *The psychology of the child*. 1969.

[16] K. A. Williams and A. Cavallo, *Reasoning Ability, Meaningful Learning, and Students' Understanding of Physics Concepts*. Journal of College Science Teaching, 1995.

[17] H. Ishii and B. Ullmer, "Tangible bits," presented at the the SIGCHI Conference, New York, New York, USA, 1997, pp. 234–241.

[18] T. Turchi, A. Malizia, and A. Dix, "Fostering the adoption of Pervasive Displays in public spaces using tangible End-User Programming," presented at the 2015 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), 2015, pp. 37–45.